Some Preliminaries
For efficiency, the graphics card will render objects as triangles

Any polyhedron can be represented by triangles

Other 3D shapes can be approximated by triangles
A Dolphin

Double Buffering

- Don’t want to draw objects directly to the screen
- The screen could update before a new frame has been completely drawn
- Instead, draw next frame to a buffer and swap buffers when complete.
Direct3D 11 Pipeline

Source: Microsoft
In order to avoid rendering vertices that will not be displayed in the final image, DirectX performs ‘culling’

- Triangles facing away from the camera will be culled and not rendered
- By default, DirectX performs ‘Counter-Clockwise culling’
- Triangles with vertices in a counter-clockwise order are not rendered
- The order of vertices is therefore important
- Left hand rule
A Windows 8 SharpDX App

Based on the Toolkit – MiniCube
App.xaml

- The App definition
- Specifies the namespace you will be working in
- Specifies application resources
**App.xaml.cs**

- The first file loaded by your App
- Sealed partial class `App : Application`
- Extends the `Application` class, allowing your program to interact with Windows (e.g. receive events)

```csharp
public App()
{
    this.InitializeComponent();
    this.Suspending += OnSuspending;
}
```

- Initializes XAML elements
- Registers the `OnSuspending` function as an event handler for the `Suspending` event
OnLaunched

- protected override void OnLaunched(...)  
  var swapChainPanel = new MainPage()  
  Window.Current.Content = swapChainPanel  
  Window.Current.Activate();

- Overriden from the Application class
- Loads the state if necessary from a suspended execution
- Creates the XAML element we will be drawing to (MainPage)
- Sets MainPage as our current Window
private void OnSuspending(...) {
    var deferral = e.SuspendingOperation.GetDeferral
    // Do Something
    deferral.Complete
}

- Defers the suspending operation until your code has had a chance to complete
- Use this function to save state information as necessary.
MainPage.xaml

- Contains (Modern Style) XAML elements, which are overlayed onto your rendered scene
- Can use these elements for user interaction
  - Buttons
  - Sliders
  - Text
  - Etc...
- Contains one key element:
  SwapChainBackgroundPanel
- This is the collection of buffers that will be rendered to
- More on this in later labs
public MainPage()
    InitializeComponent();
    game = new MiniCubeGame();
    game.Run(this);

- Initializes the MainPage XAML elements
- Creates the new Game
- Runs the game
- Also use this class to specify input / output functions for XAML elements rendered on MainPage – More on this in later labs
Public class MiniCubeGame : Game
private GraphicsDeviceManager graphicsDeviceManager;
private BasicEffect basicEffect;
private buffer<VertexPositionColor> vertices;
private VertexInputLayout inputLayout

Extends the toolkit Game class, which contains many helpful functions
Sets up device, context, etc...
LoadContent

- Protected override void LoadContent()
- Called on the creation of the Graphics Device
- Sets up initial resources such as models and effects
BasicEffect

basicEffect = ToDisposeContent(new BasicEffect(GraphicsDevice) {
    VertexColorEnabled = true,
    View = Matrix.LookAtLH(new Vector3(0, 0, -5), new Vector3(0, 0, 0), Vector3.UnitY),
    Projection = Matrix.PerspectiveFovLH((float)Math.PI / 4.0f,
        (float)GraphicsDevice.BackBuffer.Width / (float)GraphicsDevice.BackBuffer.Height, 0.1f, 100.0f),
    World = Matrix.Identity });

BasicEffects model transformations, effects, texturing, lighting, etc... That will be applied to objects
Vertex Definitions

- vertices = ToDisposeContent(Buffer.Vertex.New(GraphicsDevice, new[] { 
  new VertexPositionColor(new Vector3(-1.0f, -1.0f, -1.0f), Color.Orange),
  new VertexPositionColor(new Vector3(-1.0f, 1.0f, -1.0f), Color.Orange), ...

- Similar to the definitions in Labs 1 & 2
- Note that the vertex definitions here are typed, not just a series of floats
- This allows the input layout to be extracted from the vertex definition without needing to be specified explicitly
InputLayout

- inputLayout = VertexInputLayout.FromBuffer(0, vertices);
- Specifies the meaning of the vertex definitions
- Avoids the need to specify explicitly as was done in the labs:
  - layout = new InputLayout(d3dDevice, vertexShaderByteCode, new[]
    { new InputElement("POSITION", 0, Format.R32G32B32A32_Float, 0, 0),
      new InputElement("COLOR", 0, Format.R32G32B32A32_Float, 16, 0) });
Update

- protected override void Update(GameTime gameTime)
- Called 60 times per second (unless you change this)
- Performs necessary game calculations
protected override void Draw(GameTime gameTime)
{
    GraphicsDevice.Clear(Color.CornflowerBlue);
    GraphicsDevice.SetVertexBuffer(vertices);
    GraphicsDevice.SetVertexInputLayout(inputLayout);
    basicEffect.CurrentTechnique.Passes[0].Apply();
    GraphicsDevice.Draw(PrimitiveType.TriangleList,
                        vertices.ElementCount);
    base.Draw(gameTime);
}

Called on each frame
Clears the screen
Sets the vertex buffer & input layout
Applies the basic effect
Draws the scene
Finding out more

- F12
- http://sharpxdx.org/documentation
- https://github.com/sharpxdx/SharpDX
- Anything on XNA (use with care)